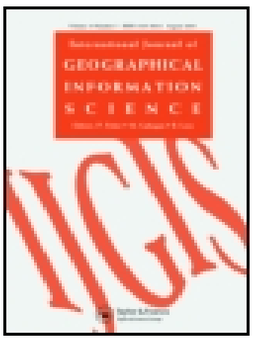

International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: http://www.tandfonline.com/loi/tgis20

A continuous deployment-based approach for the collaborative creation, maintenance, testing and deployment of CityGML models

Iñaki Prieto, Jose Luis Izkara & Rubén Béjar

To cite this article: Iñaki Prieto, Jose Luis Izkara & Rubén Béjar (2017): A continuous deployment-based approach for the collaborative creation, maintenance, testing and deployment of CityGML models, International Journal of Geographical Information Science, DOI: 10.1080/13658816.2017.1393543

To link to this article: http://dx.doi.org/10.1080/13658816.2017.1393543

Published online: 26 Oct 2017.

INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE, 2017
https://doi.org/10.1080/13658816.2017.1393543

Taylor & Francis
Taylor & Francis Group

ARTICLE

A continuous deployment-based approach for the collaborative creation, maintenance, testing and deployment of CityGML models

Iñaki Prieto [a], Jose Luis Izkara [a] and Rubén Béjar [b]

[a]Sustainable Construction Division, Tecnalia Research & Innovation, Derio, Spain; [b]Aragon Institute of Engineering Research, Universidad Zaragoza, Zaragoza, Spain

ABSTRACT

Georeferenced 3D models are an increasingly common choice to store and display urban data in many application areas. CityGML is an open and standardized data model, and exchange format that provides common semantics for 3D city entities and their relations and one of the most common options for this kind of information. Currently, creating and maintaining CityGML models is costly and difficult. This is in part because both the creation of the geometries and the semantic annotation can be complex processes that require at least some manual work. In fact, many publicly available CityGML models have errors. This paper proposes a method to facilitate the regular maintenance of correct city models in CityGML. This method is based on the continuous deployment strategy and tools used in software development, but adapted to the problem of creating, maintaining and deploying CityGML models, even when several people are working on them at the same time. The method requires designing and implementing CityGML deployment pipelines. These pipelines are automatic implementations of the process of building, testing and deploying CityGML models. These pipelines must be run by the maintainers of the models when they make changes that are intended to be shared with others. The pipelines execute increasingly complex automatic tests in order to detect errors as soon as possible, and can even automate the deployment step, where the CityGML models are made available to their end users. In order to demonstrate the feasibility of this method, and as an example of its application, a CityGML deployment pipeline has been developed for an example scenario where three actors maintain the same city model. This scenario is representative of the kind of problems that this method intends to solve, and it is based on real work in progress. The main benefits of this method are the automation of model testing, every change to the model is tested in a repeatable way; the automation of the model deployment, every change to the model can reach its end users as fast as possible; the systematic approach to integrating changes made by different people working together on the models, including the possibility of keeping parallel versions with a common core; an automatic record of every change made to the models (who did what and when) and the possibility of undoing some of those changes at any time.

ARTICLE HISTORY
Received 14 June 2017
Accepted 13 October 2017

KEYWORDS
CityGML; collaborative edition; continuous deployment; 3D city model; automated testing

CONTACT Iñaki Prieto inaki.prieto@tecnalia.com

1. Introduction

Georeferenced 3D models are an increasingly accepted solution for storing and displaying urban data, and the CityGML standard is among the best options for this. It combines 3D information and semantic information in a single data model, facilitating the use and interoperability between models (Gröger *et al*. 2012). This kind of data model permits representing, handling and managing urban data that can be used in different applications such as disaster management, urban planning, traffic planning, security, telecommunications, navigation or tourism (Biljecki *et al*. 2015b).

Creating high quality CityGML models is costly in regard to time and money (Döllner *et al*. 2006). The geometries can be generated with different data acquisition technologies such as 3D laser scanning, image-based modeling and computer-aided design (Limp *et al*. 2010). This geometric generation can be performed in a semi-automatic way but it usually requires at least some manual editing (Malamboa and Hahnb 2010, Cao *et al*. 2017). The main limitation of the models generated with these methods is that they lack semantic information. Some authors consider that the main bottleneck to speeding up the adoption of CityGML as an open standard is the difficulty of annotating 3D models with semantics, which requires at least some manual labor (Kang and Hong 2015).

Another important challenge regarding 3D city models is simplifying their maintenance. The models should be kept up-to-date as the real cities change, and should be extended when new information is available. If making changes to the models were simpler, the generation of 'what-if' scenarios and the analysis of alternatives, such as finding out the consequences of different changes in a certain neighborhood, would be more common and less challenging than they are now (Ohori *et al*. 2015).

Even when CityGML models exist and are actively maintained, there is still another major issue: most of the existing models have geometric and semantic errors (Biljecki *et al*. 2016). For example, in a freely accessible CityGML model of the city of Rotterdam there are errors such as wrong orientations, intersection of surfaces, missing surfaces or non-planar surfaces, in 90% of the city objects (Peters *et al*. 2014).

When the maintenance of CityGML models cannot be done by a single person, for instance when they become larger and more complex, or when the person responsible for this has too many other tasks, it is necessary to consider their collaborative creation and maintenance. This may bring several benefits, but also some problems. These benefits would include faster results, thanks to the parallelization of the effort, and the possibility of having different professionals with different expertise working on different aspects of the same model. Collaborative maintenance also opens the door to developing participatory urbanism initiatives (Grey *et al*. 2017; Dyer *et al*. 2017) and makes it possible to consider the participation of volunteers in the maintenance of the models in the future. However, this would involve some additional problems, such as the integration of changes that may be in conflict with each other and the validation of those changes.

This paper presents a solution to facilitate the regular maintenance of 3D city models in CityGML. This solution is based on the continuous deployment (CD) strategy and tools used in software development, and adapts them to the problem of creating, maintaining and deploying correct CityGML models. The paper shows how this solution contributes to solving the challenges and problems identified before: (1) it reduces manual labor by automating processes, (2) it facilitates the collaborative maintenance of the models by

integrating a version control system (VCS) in the workflow, and (3) it reduces geometric and semantic errors by the systematic and frequent execution of automatic tests.

The rest of the paper is structured as follows: first, some related work is reviewed in Section 2. The proposed solution, a method to automate the deployment of correct 3D city models, is presented in Section 3. Section 4 describes an example scenario used to test the proposed solution. Section 5 discusses the rationale behind some of the adopted solutions. Finally, the paper ends with some conclusions and future work recommendations.

2. Related work

Regarding the collaborative creation and maintenance of georeferenced city models, the OpenStreetMap (OSM) project is currently an excellent example of this for 2D data (Budhathoki and Haythornthwaite 2012). OSM has its own data model and tools, with integrated, ad hoc support for multiuser edition and version control. In the case of 3D cities, there have been a number of proposals. For instance, there is a methodology for the collaborative, interactive development of 3D building models that considers walls, roofs, doors and windows as structural elements (Abbasi and Malek 2015). Other authors have enhanced the OSM data in order to support a third dimension. For example, the OSM-3D platform has been developed to provide a web-based interactive 3D view of this data (Goetz and Zipf 2012, Goetz 2012). However, the 3D in OSM is based on extrusion, so complex geometries cannot be represented. Another solution is 3D Repo, which is a pioneering open-source version control framework that enables coordinated management of large-scale engineering 3D data over the Internet, though it does not currently support CityGML (Scully *et al*. 2015).

The architecture, engineering and construction industry has developed several methods and building information modeling (BIM) processes, to facilitate collaboration among the actors involved in the development process of built assets. As an example, the aim of ISO 29481 is to facilitate interoperability between software applications used during all stages of the life cycle of construction works (ISO 2010). The main focus of BIM is in the building scale (not the city scale) and in the building construction processes more than in the modeling processes.

Some data models need to be kept up-to-date to be fit for their purpose. In the case of 3D city models, their obsolescence factors have been studied and some obsolescence–prevention strategies have been proposed. These strategies include considering factors such as file formats, data interoperability, accessibility or usability.

In addition, the regular maintenance of a city model would create a history of changes that can be useful to analyze the temporal evolution of the city. Nevertheless, the CityGML standard currently does not provide us with a way to store this kind of information, because only the current state values are included (Morel and Gesquière 2014). An alternative is to extend CityGML to allow for versions of the entire model, or of some specific city elements, to represent different planning alternatives (Chaturvedi *et al*. 2015). There is also a modeling approach and an implementation for supporting the management of versions and history within CityGML (Chaturvedi *et al*. 2015). Another related problem is to keep the changes under control as the models evolve. Changes are fully controlled if it is well-known

who has changed what, when and for what reason. This issue, in addition to versioning, has been identified since the beginning of CityGML (Gröger *et al*. 2005), but it has not yet been considered in the standard.

Regarding the correctness of CityGML models, there are a good number of proposals to facilitate the detection of errors and the healing of these models. The detection of geometric errors in 3D city models is necessary to guarantee the output of processing or manipulation operations such as the calculation of building volumes, adjoining walls or solar radiation (Sindram *et al*. 2016, Biljecki *et al*. 2015a). The SIG 3D Quality Working Group has defined guidelines with modeling recommendations in order to avoid errors in the generation of CityGML models (Gröger and Coors 2011). Even when the data are standard-compliant, some rules for the validation of geometric-semantical consistency are still needed (Wagner *et al*. 2013, Zhao *et al*. 2014). Alam *et al*. (2013) have developed a tool to repair the errors detected in the CityGML models automatically. Val3dity is a tool to validate 3D primitives according to the international standard ISO 19107 (Ledoux *et al*. 2013). CityDoctor implements methods and metrics for analysis, testing and correction of syntax, geometry and semantics of virtual 3D city models (Coors and Krämer 2011). Currently, none of these tools can detect and repair all the possible errors within a CityGML model. The CityGML Quality Interoperability Experiment provided a set of recommendations on conformance requirements related to quality-checking tools and a validation workflow (Wagner and Ledoux 2016).

3. CD of CityGML models

This section describes a method to automate the validation and deployment of CityGML models, which also facilitates the collaborative maintenance of these models by several people. Subsections 3.1 and 3.2 are completely technology-independent and describe the entire solution. This method is based on the CD strategy used in software development, and applying it to a CityGML model mainly consists of carrying out two main tasks:

(1) Designing and implementing a deployment pipeline for that model. This, in short, will consist of a sequence of linked, automatic stages that transform its input, a modified CityGML model committed to a VCS repository, to a validated and deployed CityGML model. These pipelines could be implemented ad hoc, e.g. as scripts in any programming language, but leveraging specialized tools provides a good starting point. This just needs to be done once per city model, and parts of the pipeline may be reused.
(2) Executing that pipeline when a set of changes to a CityGML model that we want to share is committed. The frequency depends on the number and complexity of the changes, and also on the number of people working at the same time on the model, but it would be common to commit changes and execute the pipeline at least once per day, in order to minimize merge conflicts when simultaneous changes are made by several maintainers.

The rest of this section describes how to design and implement these deployment pipelines. As the proposed method is based on CD, we start by briefly describing this software development strategy.

3.1. *Continuous deployment*

CD is a methodology used to systematically control and automate the testing and deployment of software and services (Humble and Farley 2010). Although CD is mainly used in software development, there are some examples of its application in the deployment of other artifacts, such as documents (Gentle 2015; Vakharia 2015).

CD is based on the concept of a deployment pipeline, which is an automatic implementation of the process of building, testing and deploying a software system. A deployment pipeline usually has several stages (although it may have only one), each one slower than the previous ones but providing more confidence in the correctness of the software being deployed. This increasing confidence is provided by the automatic execution of increasingly complex tests. The implementation of a deployment pipeline requires at least a VCS repository where the changes to the code are committed by its developers.

A basic deployment pipeline starts with a commit stage that takes some changes committed to a source code repository, compiles and assembles the binaries (e.g. the executables) and runs some simple tests. If this fails, the programmer is informed, and if it succeeds, an acceptance stage is run next, where a test environment is configured, the binaries are deployed there and some acceptance tests are executed. Once that stage succeeds, a final production stage that configures the production server and deploys the binaries there for the end users can be run.

Other stages can also be added, such as a capacity stage to test the performance of the software being deployed, or a user acceptance stage, where other user-facing tests, maybe even manual ones, are run.

3.2. *CityGML deployment pipelines*

The proposal in this paper requires designing and implementing deployment pipelines for the CityGML models being created, maintained and deployed. These pipelines will follow a similar structure to the deployment pipelines for software systems described in the previous section, but many of the tasks carried out will be different because the problem solved is different. As with software deployment pipelines, the CityGML pipelines will have to be tailored to each city model, but they will share a common structure with up to four stages as shown in Figure 1:

- **Commit Stage**: It checks if a CityGML file is structurally correct and standard-compliant. This stage takes a CityGML file committed to the VCS as input, it checks if the file contains structural errors or nonconformances related to the CityGML standard, and also calculates some statistics on the elements contained in the file. If the CityGML file is structurally correct, this stage outputs a number of metrics and statistics on that file. If the input was an incorrect file, a list of errors is provided as a result.
- **Acceptance Stage**: The aim of this stage is to deploy the city model in the same format required by the end user-oriented deployment, in a test environment where

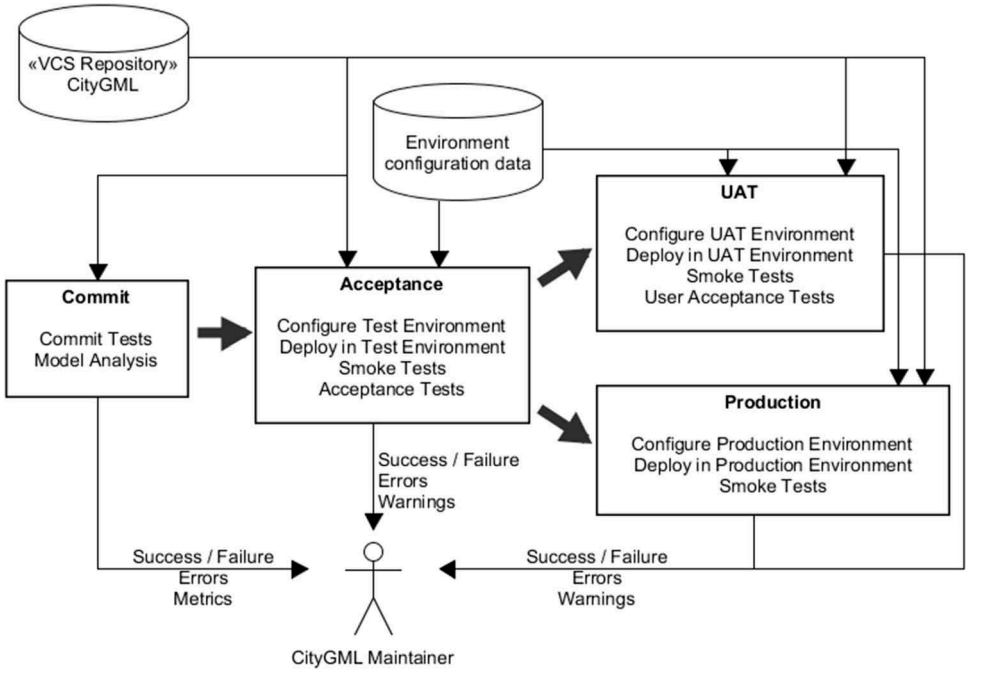

Figure 1. CityGML deployment pipeline.

a number of acceptance tests are performed to ensure that the model meets the user requirements in an environment similar to the production one. This stage takes a structurally correct CityGML file as input. The tests performed at this stage are focused on the acceptance criteria, which are based on the user requirements. This stage produces a list of errors and warnings related to these tests. The more acceptance criteria are defined and implemented as automated tests in this stage, the more confidence we may have in the fact that the CityGML model will fulfill the user requirements in the production environment.

- **User Acceptance Test (UAT) Stage**: There may be some acceptance tests that have to be done manually because they have not yet been automated or because they cannot be automated (e.g. a visual exploration of a city model can help to detect some inconsistencies). This stage ensures that these manual UATs are done. This stage deploys a CityGML file which has been checked by the commit and the acceptance stages in a user acceptance testing environment. A tester will be informed that there are manual tests waiting to be performed and the pipeline will be paused until these tests are done and marked as 'success' or 'failure'. After this the stage will be completed. The output of this stage is a report with the result of the stage and, in case of failure, a list of errors found and comments provided by the tester.
- **Production Stage**: The aim of this stage is to deploy the city model, in the format required by the end users, in a production environment. This stage takes a CityGML file that has passed through the commit, acceptance and UAT stages as input; configures a production environment; and deploys the city model there and runs some smoke tests, which are very quick and simple tests used to check that a

process has been carried out without major failures. A report with the results of the process will be provided as a result of the stage and, in case of success, the CityGML file will be available to the end users. The term deployment is used here in a broad sense. This may mean to copy it to a directory accessible via a web server, to integrate it with a web application that makes use of it, maybe through a web service interface, or even to pack it as a resource inside a mobile app before publishing it on an app market.

The tasks carried out in these stages are explained in the following sections.

3.2.1. *Commit stage*

3.2.1.1. *Commit tests.* The validation of the structural correctness of the CityGML file requires: (1) XML schema validation (XML, XSD and CityGML version), (2) geometric validation to check that the representation of the geometric information is compliant with the GML standard (that polygons are closed, consecutive points are not repeated and others), and (3) semantic validation, which includes checking the code lists for enumerative attributes.

3.2.1.2. *Model analysis.* This task will calculate various descriptive statistics from the CityGML file. These metrics will include the number of elements in the model, the elements for each Level of Detail (LoD) and the number of those elements including a basic set of attributes (e.g. class, function and usage).

3.2.2. *Acceptance stage*

3.2.2.1. *Configure test environment.* The first task of this stage is to set up the environment in which to conduct the acceptance tests. The environment configuration data should include the required configuration settings for software components like the database server and the web server.

3.2.2.2. *Deploy in test environment.* This task deploys the city model, in the same format required by the end user-oriented deployment. For example, a database or a file system could both be options to be used for the deployment.

3.2.2.3. *Smoke tests.* Smoke tests are very quick and simple tests used to check that a process has been carried out without major failures. The ones here check that the deployment process in the test environment has finished without critical errors. For instance, in the case of database storage, a basic request for data retrieval could be a sufficient smoke test. If the deployment is performed in a file-based storage, the test could check whether the file has been correctly uploaded into the repository (e.g. by checking that the size of the uploaded file is correct).

3.2.2.4. *Acceptance tests.* This task will check that the city model meets the specifications defined by the application and the customer, expressed as acceptance criteria related to the requirements, so it is problem-domain related, and that it works properly in an environment similar to the one in production. For example, this task will typically check that the model contains all the required semantic properties (for instance, the area

and orientation of facades and adjoining walls). Within this task, the correct hierarchy of the buildings can be also checked. As an example of this, the rules in a certain domain could be that buildings contain parcel information and geometry in LoD0 and LoD1, and that building parts contain real building information and geometry in LoD2. Other possible tests here could be to ensure that the geometry height and the height given in the measuredHeight parameter are similar, or that walls are represented by vertical surfaces and grounds and roofs are surfaces that tend to be horizontal.

3.2.3. *UAT stage*

3.2.3.1. ***Configure, deploy and run smoke tests in the UAT environment.*** The actions to be performed in these tasks are like those explained in the Acceptance Stage, but they are performed in the UAT environment.

3.2.3.2. ***User acceptance tests.*** This task will check that the deployed city model is valid from the user perspective, considering elements that are difficult or impossible to express as automatically testable acceptance criteria, so they have to be done manually. First, the model can be checked as an isolated element. Even if the geometric information is correct according to the standard, the geometry could be invalid (e.g. courtyards are not represented or have an incorrect texture mapping). Once the model is checked as an isolated element, its integration in the final environment can be checked as well, considering its geolocation and orientation, for instance. A human tester will be informed that there are manual tests waiting to be performed, and the pipeline will be paused until these tests are done and marked as success or failure. These tests can be done by following a script, but exploratory testing (an experimented tester is given the opportunity to explore the model without a predefined script) can be a valuable tool to discover unexpected problems.

3.2.4 *Production stage*

3.2.4.1. ***Configure, deploy and run smoke tests in the production environment.*** The actions to be performed in these tasks are like those explained in the Acceptance Stage, but they are performed in the Production environment.

3.3. *Implementation of CityGML deployment pipelines*

This section discusses a possible implementation of CityGML deployment pipelines like those described in the previous section. The implementation of software is technology-dependent, but the design described in the previous sections is not, so it could be implemented with different technologies than those described here. The main structure of these pipelines can be supported by any VCS and any CD tool. These tools are advanced, well-documented and there are many to choose from. Being able to make use of them is one of the advantages of having based our proposal on software deployment pipelines. The other necessary tools are those that provide the functionality required by the tasks in the stages of the pipelines, most of them specifically related to CityGML. These tools are the rough equivalent to the compilers, linkers, code analyzers or testing frameworks that a software deployment pipeline would use. Some of these CityGML tools have been developed by us, while others are adaptations of existing ones.

Regarding the pipeline structure, a VCS is required first. VCSs record changes to files over time so it is possible to recall specific versions later. Nowadays, distributed version control systems (DVCS) like Git or Mercurial are increasingly popular and provide a whole set of novel capabilities. Using DVCS, developers can work on local copies of the repositories, enabling them to work offline while still retaining the full project history; they can cheaply create and merge branches; and they can share only their chosen sets of changes.

Besides a VCS, a CD tool is necessary to implement the pipeline structure. There are many to choose from, being some of the most common Jenkins, Team City, Cruise Control, Bamboo and GoCD.

Table 1 lists a number of tools for the pipeline implementation, including a short description, their inputs and outputs and indicating if they have been specifically developed (D) or are third-party tools (TP). Most of them are specifically developed for CityGML. These tools have been implemented in Java as simple REST web services that can be easily called from any CD tool, and are not specific of a certain CityGML model or pipeline, so they can be reused. These tools are offered as web services, so they could be implemented in other programming languages. These tools will typically be reusable, but tools that are specific for a certain model or special problem could be developed. The currently implemented set of tools is meant to provide a number of functionalities that allow for implementing a representative CityGML deployment pipeline in order to validate the proposal in this paper.

A number of third-party libraries and tools have been used, as shown in the Table 1. Some of these are specifically developed for CityGML, while others are more generic and have been adapted to be used in a CityGML deployment pipeline by wrapping them as REST web services in Java.

4. Example scenario

This section applies the solution proposed in this paper, based on CityGML deployment pipelines, to a scenario that is representative of the kind of problems that this solution aims to solve. This scenario serves both as a test of the feasibility of the proposal and as an illustrative example of its possibilities. It is based on real data and on work in progress related to the energy retrofitting of buildings (Prieto *et al*. 2015).

In this scenario, several actors collaborate to maintain a city model that is used, possibly among other things, to identify the most appropriate solutions for the energy retrofitting of a city neighborhood. The scenario is located in the city of Santiago de Compostela (Spain), for which a CityGML file has been created. This CityGML model is updated frequently and it is important that this model is correct at any time. The maintenance of the CityGML model will be performed by the following actors:

- **City hall architect**: The city hall architect is in charge of keeping the main elements of the CityGML up-to-date. He will periodically update the CityGML models with data provided by other institutions (e.g. the cadaster or the national mapping agency). The city hall is only concerned with basic semantic properties and the geometry of the city elements at low levels of detail (LoD0, LoD1 and LoD2).

Table 1. Developed and third-party tools.

Name	Description	Input	Output	D/TP
Commit Stage				
CityGML version checker	CityGML module namespaces are checked to find out if they refer to CityGML version 2.0.0.	CityGML file	Version is valid or error list	D
CityGML geometric checker	Currently, for every surface, this tool checks that the first and the last points are equal, and that there are not any consecutive repeated points.	CityGML file	Geometrically valid or error list	D
CityGML semantic checker	This tool validates the model against the defined CityGML code lists.	CityGML file	CityGML code lists are correct or error list	D
CityGML statistics generator	This tool calculates, for all the buildings in the CityGML model, how many of them have each of the following elements filled in: class, function, year of construction, roofType, measuredHeight, storeysAbove and storeysBelow.	CityGML file	Metrics or error list	D
Java XML Parsers	The CityGML file is parsed to find out if it is well-formed.	CityGML file	CityGML file is well-formed or error list	TP
Java SchemaFactory	The CityGML file is parsed to find out if it is schema valid.	CityGML file	CityGML file is schema valid or error list	TP
Acceptance Stage				
Name	Description	Input	Output	D/TP
Facade checker	This tool checks that each facade has its area and orientation parameters calculated.	CityGML file	Parameters are calculated or error list	D
UAT Stage				
Name	Description	Input	Output	D/TP
Cesium – Integration in the real environment checker	The CityGML model is visualized in a 3D globe and maps web application.	URL of the CityGML file exported to KML	CityGML is visualized in Cesium.[a]	TP
Acceptance, UAT, Production Stage				
Name	Description	Input	Output	D/TP
WFS request generator	In order to smoke test that a WFS has been correctly set up, this tool sends a GetCapabilities request to it and checks the service has been correctly set up.	URL of the WFS	WFS set up is correct or error list	D
3DCityDB – Database structure generator	Creates the SQL schema required to store a CityGML model.	3DCityDB PostgreSQL script + coordinate reference system + connection and configuration details (e.g. host, port, user, password)	Database generated with CityGML structure or error list	TP
3DCityDB – CityGML to DB importer	The CityGML model is imported to the database. Previously, the content of this database is deleted.	CityGML file + connection and configuration details (e.g. host, port, user, password)	CityGML file is stored in the database or error list	TP
Deegree	The WFS is set up with the configuration of the database, features to make available and the service.	Project name + connection and configuration details (e.g. host, port, user, password)	The WFS is set up or error list	TP

[a]Cesium is a 3D globe and web maps application. http://cesiumjs.org/.

- **GIS technician** (external to the city hall): She is responsible for completing and improving the LoD of the city model by adding missing elements and by improving the LoD of the existing elements up to LoD3 (which includes doors and windows).
- **Energy consultant**: He is in charge of completing the semantic information of the CityGML model with some extensions. To get the information, he performs field work every day, mainly visual inspections of buildings, and edits the CityGML. This information can be applicable to the building itself or to its elements (e.g. year of construction, building function, main building material, installations or window percentage and materials).

4.1. *Configuration*

Figure 2 shows the configuration used in the example scenario. The setup of the scenario is not necessarily done by CityGML content developers, because it requires some experience with system administration. There is a shared Git repository hosted in GitHub where the latest correct version of the CityGML file will be shared. This is the reference copy of the CityGML model. It will be copied to the production server (in XML format and as a PostGIS database) for its exploitation with every successful execution of the full pipeline. All the actors have read and write permissions to this repository (i.e. they follow the so-called centralized workflow in Git (Chacon and Straub 2014)). If there were information that could not be shared among all the actors, it would be necessary to create a different Git repository with different permissions and to ensure that changes in both repositories are properly synchronized. They all have a local clone of this shared repository, where they pull from the shared repository and commit their changes as they work. In order to test that these

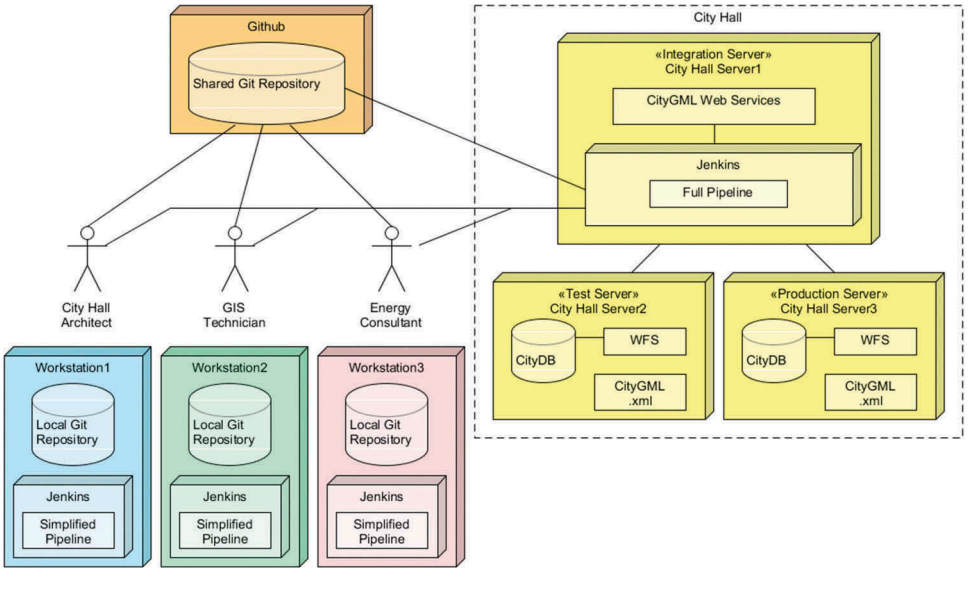

Figure 2. Configuration of the example scenario.

changes have not broken any commit tests before sharing them with the others, they all have a Jenkins CI server in their local machines where they run a simplified version of the shared deployment pipeline, which only has the Commit Stage. Once a commit passes through this shortened local pipeline successfully, they may decide to share it through the GitHub repository by pushing it.

The GitHub repository is linked with a full deployment pipeline in a Jenkins CI server instance hosted in a computer in the city hall (the integration server). The integration server has access to a test server, where the tests of the acceptance and UAT stages are run, and to the production server, where the new version of the CityGML file will be deployed, and thus made available to the citizens if it passes through the full pipeline. These servers have a PostgreSQL database with a PostGIS extension and a WFS service, which are used to publish the CityGML model. Moreover, the CityGML is also deployed as a downloadable file for users who just need this. The full pipeline is automatically triggered by every push to the shared repository. Table 2 lists the tools used in the full pipeline (explained in Section 3.3)

The implementation of the deployment pipeline has been done using Jenkins, which is an open-source continuous integration application that also supports CD (Berg 2015). The main functionality provided by Jenkins is to execute a list of steps that can be executed based on time or events. A large number of plugins is available to extend its core functionality. For this example, a Git Plugin and an HTTP Request Plugin have been used. The full code of the Jenkins script used for this example is available at https://github.com/Tecnalia-CityGML/CityGML-Deployment-Pipeline-Script.

The services listed in Table 2 have been developed for this experiment by hard-coding the environment configuration data, so they are currently tied to our test and production environments. A proper, reusable implementation would have this data as a parameter of the service.

Regarding the UAT Stage, the CityGML is downloaded from the test server and visualized using Cesium. Figure 3 shows the graphical user interface that Jenkins uses to wait for the tester input on the left, and on the right, the CityGML model as shown in Cesium.

4.2. *Execution and results*

In this scenario, the three actors make some changes to the model. When they update the CityGML model with new geometric and/or semantic information, they check the model through their local pipelines. Once it passes the local pipeline successfully, the

Table 2. Tasks of each stage within the validation pipeline.

Stage	Tools
Commit Stage	Java XML Parsers. Java SchemaFactory. CityGML geometric checker. CityGML semantic checker. CityGML statistics generator.
Acceptance Stage	3DCityDB – Database structure generator. 3DCityDB – CityGML to DB importer. CityGML WFS deployer. Deegree – WFS request generator. Facade checker.
UAT Stage	The deployment and smoke tests are like those in the Acceptance Stage. Also Cesium integration in the real environment test is performed. A tester interaction is needed at this step, so the pipeline will be paused until the tester gives some feedback.
Production Stage	The same tools developed for the Acceptance Stage, but performed in the Production environment.

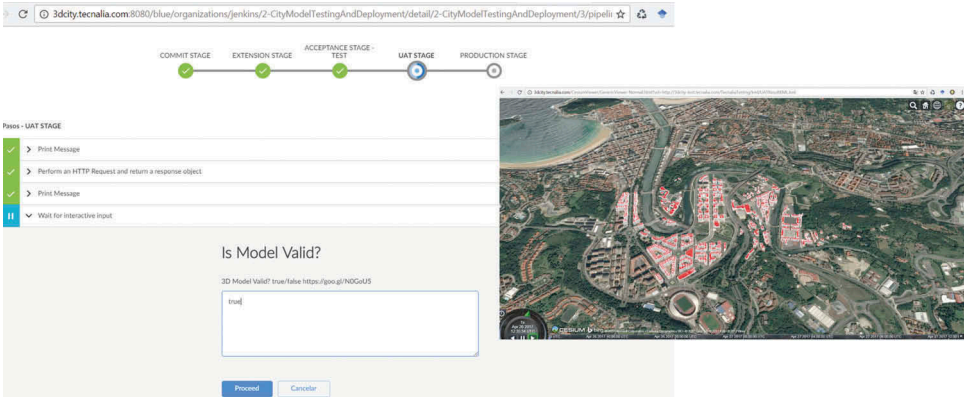

Figure 3. UAT visualization in Jenkins and Cesium.

changes are pushed to the shared repository and the full pipeline is triggered. In the following example scenario, five successful deployments of the city model are done, and five errors (geometric and semantic errors that occur in various LoDs) are automatically detected before they reach the end users.

The diagram in Figure 4 shows the detailed sequence of actions carried out by the three actors in the example scenario:

- First, the city hall architect creates the city model with 300 buildings in LoD2 (1). The local and shared pipelines are successful but some warnings are notified to point out that there are some facades with areas smaller than 1 m^2, so they probably have to be fixed.
- The energy consultant pulls the city model (1) and adds the semantic information of two buildings (2). His local pipeline fails because the function code types of the buildings are not correct. The energy consultant corrects those building code types (3) and after that, both the local and the shared pipelines finish successfully.
- Some days later, the GIS technician pulls the city model (3) and completes the 300 buildings including their LoD0 and LoD1 (4). However, her local pipeline does not finish successfully because there are errors in some geometries. The GIS technician starts to fix the geometric errors. Meanwhile, the energy consultant pulls the city model (3) and completes the semantic information of another building (5). Both the local and the shared pipelines finish successfully. The GIS technician finishes solving the geometric errors (6) and her local pipeline works successfully. However, when she tries to push the city model to the shared repository a conflict occurs because another set of changes have been pushed in the meanwhile (5), so the GIS technician has been working locally with an old version. She pulls the current city model (5), merges that model with the latest changes in her machine (7) and then both pipelines work successfully.
- After that, the city hall architect pulls the city model (7) and completes it by adding the LoD0, LoD1 and LoD2 of 263 buildings (8). The shared pipeline fails because there is a user requirement (checked by an acceptance test) which requires that

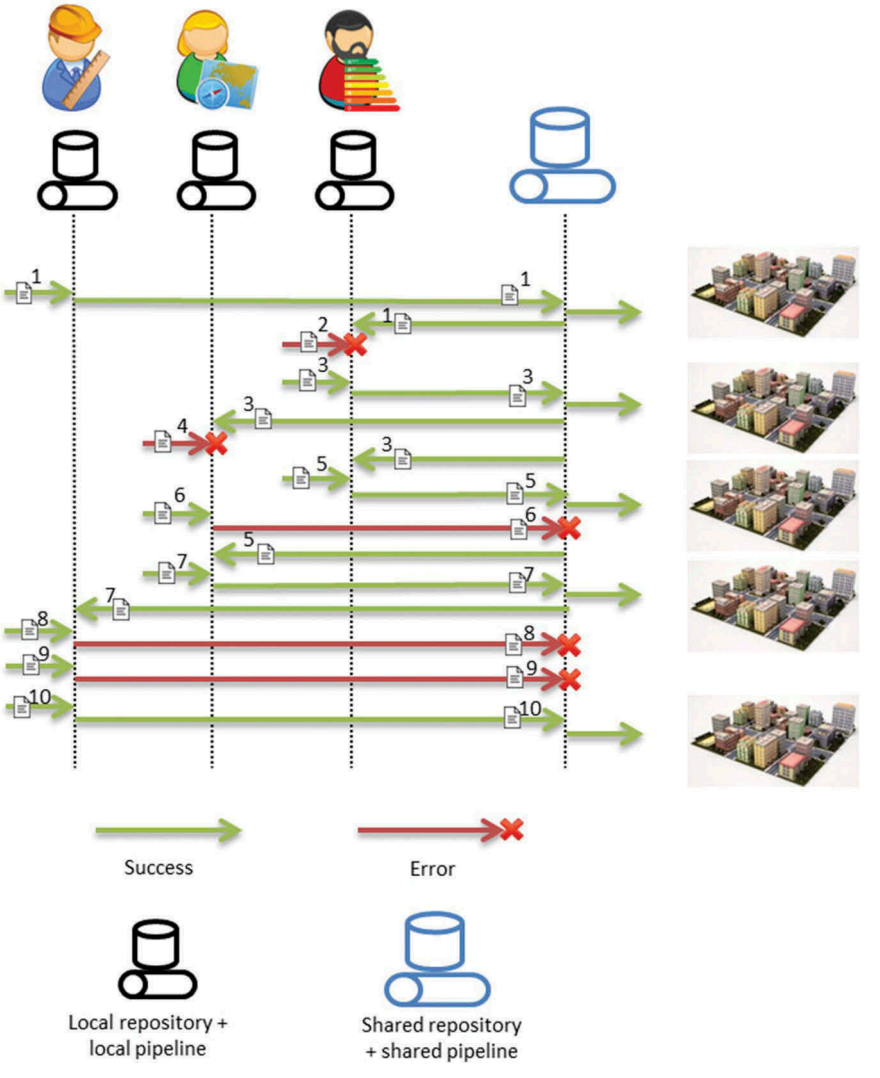

Figure 4. Main interactions in the example scenario.

every facade must have an area and an orientation. The architect corrects the errors (9) and pushes the changes to the shared repository, thus triggering the shared pipeline. In the UAT Stage of that pipeline, a tester notices that some buildings are not correctly located, so the pipeline fails at that stage. After fixing these errors (10), both pipelines pass successfully.

The Git repository used to run this example scenario is hosted in GitHub at https://github.com/Tecnalia-CityGML/CityGML-Deployment-Pipeline.git. The commit log shown in Figure 5, on the left, is taken from the master branch of that repository. The different triggers of the pipeline in Jenkins can be seen on the right side.

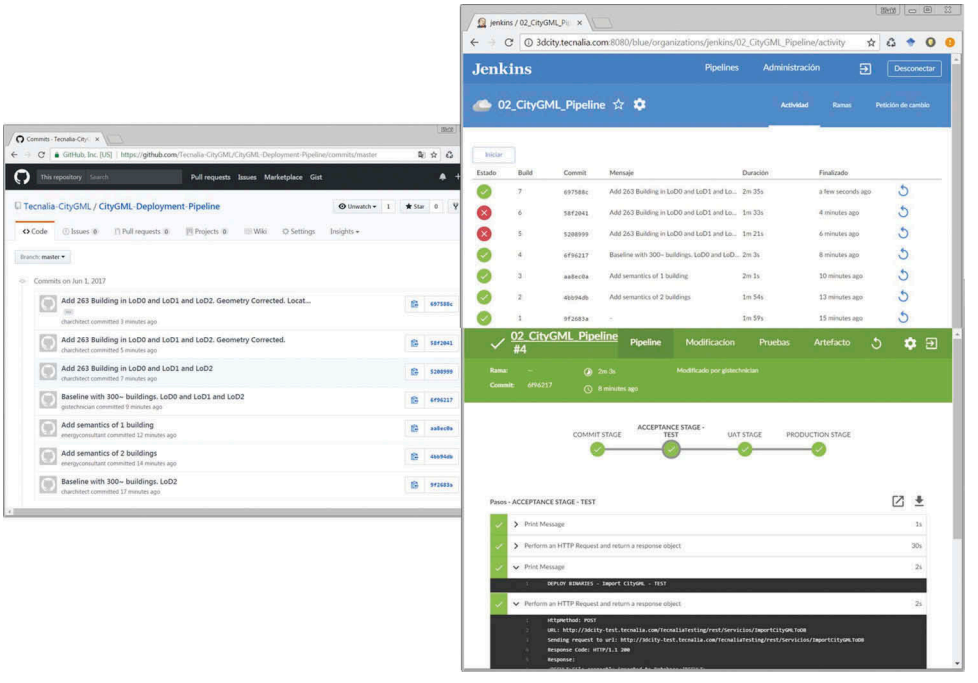

Figure 5. Different triggers of the pipeline in Jenkins.

5. Discussion

In this section, we will discuss the rationale behind some of the main decisions made to develop the proposal in this paper.

First, we have selected CityGML as the data model because we have created a number of city models for different projects, and this has allowed us to identify an issue with the maintenance and validation of those models. The CityGML standard was published 9 years ago but, as reviewed in Section 2, there are still many non-correct, i.e. containing semantic and geometric errors, CityGML models publicly available. In any case, a deployment pipeline based on a VCS and both automatic and manual tests can be applied to other complex data formats as well, which is something that can be explored in the future.

Currently, validation of the models is usually done manually. This is time-consuming and, consequently, too often skipped. Having identified manual procedures as one of the main sources of the problems identified in the maintenance and deployment of valid CityGML models, we had to choose a strategy to make these procedures automatic. Instead of designing and implementing an ad hoc solution, the field of software engineering has provided us with the CD strategy and tools. The main advantage of adopting this strategy is that we have been able to benefit from the existence of specialized and powerful tools. Another advantage is that because this strategy is becoming increasingly common in software development, it may be easier to find people who already know about it when hiring people to work on the maintenance and deployment of CityGML models.

The proposed solution includes the deployment of valid CityGML models, where deployment means 'to make the model available to its potential users'. There are not many CityGML models available now, and their deployments are typically simple (e.g. making them available through a web server), so it may seem unnecessary to automate this. However, the proposal in this paper intends to make it easier and faster to make changes to CityGML models. If changes are easier, they may be done more frequently. If they are more frequent, deployments, even simple ones, will also be more frequent and require more time, so making this part of the process automatic (instead of manual and thus error-prone) makes more sense. In addition, other more complex scenarios (e.g. cloud based deployment with dynamic load balancing, or those where software and CityGML models are packaged together) will benefit even more from the automation of the deployment.

In case 3D data with enrichments need to be shared, it would be necessary to have a public repository with public data, and one or more private repositories synchronized with the public one. In this way, users without permissions will work against the public repository (as is done now or using a fork of the repository and then sharing changes with the pull requests); while users with permissions would maintain the private repository (in which they will include private data) and the public repository synchronized.

Setting up a CD strategy requires a VCS. CD tools are very well integrated with VCS and thanks to them it is possible to have a record of changes, with information about who made what change when and for what reason, and it is also possible to go back to any previous version. Moreover, some kind of version control is required to allow for collaborative work, though this does not need to be a VCS and can be made part of the data model, for instance.

As a final advantage of our proposal, it is important to highlight that it can be adopted in an incremental way, with a VCS repository as a first step, a simple pipeline with a single Commit Stage as a second one, an extended pipeline with the Acceptance Stage as a third step and a full pipeline as the final target. All these steps are useful by themselves (i.e. implementing them provides a solution to some of the identified issues), and the former are required to set up the latter. Incremental adoption makes it less risky to try this solution in a real environment.

The interactions between the actors and the exchange of data in workflows that follow the proposed method could be expressed in different ways, for instance, with Business Process Modeling Notation (BPMN), which can be used to model generic collaborative processes (Allweyer 2016). This should be explored in organizations that already use BPMN for some of their workflows, especially if CD is integrated with some of these workflows.

This research has some limitations that need to be addressed in the future. If there is a significant number of actors, there will probably be more conflicts among their changes. This could require a different Git workflow instead of the centralized one used here. In addition, some scenarios could also require some actors to validate the changes before they go through the full pipeline. The drawback is that a manual step would thus be introduced and that may lead to bottlenecks. Moreover, the maintenance of the code of the pipelines, the associated code (e.g. for the automatic tests) and the servers is out of the scope of an average CityGML modeler, so someone with more experience in system administration will be needed. Finally, GIS desktop applications, CityGML and 3D modelers, Git and automatic testing tools are not integrated nowadays,

so they must be run as separate tools (unlike the integrated development environments for software, where code creation, testing and version control are tightly integrated).

6. Conclusions and future work

This paper proposes a method to facilitate the regular maintenance and deployment of correct 3D city models expressed in CityGML. This method consists of designing, implementing and applying a deployment pipeline for each CityGML model that needs to be created, maintained and deployed. These CityGML deployment pipelines implement the procedures and tests required to transform a set of changes to a validated, updated and deployed CityGML model, or to provide information about the problems found.

A CityGML deployment pipeline has been developed and applied to an example scenario which is both representative of the kind of problems that this solution aims to solve and based on real work in progress. It is necessary to point out that the implementations of some of the CityGML-specific tools for this example have the environment configuration data hard-coded, so they are currently tied to the test and production environments used in the experiments. There is a proper, reusable implementation of these tools currently in development that will take these data as parameters.

The tools implemented to validate this work are currently a proof-of-concept. However, the full source code of a simple pipeline has been made available, so interested readers can easily set up a basic pipeline in order to extend it and develop their own. The code can be found at https://github.com/Tecnalia-CityGML/CityGML-SimplifiedPipeline.git.

The structure for the CityGML deployment pipelines is based on the CD pipelines used in software engineering, but it has been modified to fit a different problem. With this generic structure defined, a specific pipeline for each CityGML model has to be designed and implemented. The design will choose, or add, stages and tasks depending on specific constraints and needs, some of them related to the problem domain that the CityGML model is designed for. Once designed, the specific pipelines could be implemented in any scripting language, but specific CD and VCS tools (such as Jenkins and Git, used in the example scenario) are very good choices, as they are designed to help solve this kind of problems.

One of the drawbacks of the CityGML standard is that currently it seems to be difficult to create totally correct models, as recent research shows (Biljecki *et al*. 2016). Our solution intends to make it simpler to create, maintain and deploy correct CityGML models. Hopefully, this will contribute to improve the quality and availability of CityGML models and to encourage a wider use of this standard.

As future work, our approach to the CD of 3D city models can be tried in, and adapted to, more complex scenarios. We will continue completing the pipeline with more checking tools. The use of CityGML deployment pipelines should make it easier to set up large, collaborative CityGML maintenance scenarios. For example, a city could have an infrastructure that permits the CD of the city model by the different departments of the city hall. External companies, or even volunteer citizens, could be allowed to contribute some updates and fixes to that city model. Furthermore, semi-open scenarios and more complex GIT workflows can be developed in case that certain information cannot be shared between different actors. The facilitation, detection and

integration of conflicts in simultaneous changes of CityGML models could also be considered for a future line of work.

Participatory urbanism (Isikdag and Zlatanova 2010) is another field where this proposal can be useful in the future. For example, there could be a public call to propose the design of a new park, based on the common CityGML model of the city. Different alternatives could be proposed, compared and mixed based on the flexibility provided by a DVCS and also based on the safety net against errors provided by the tests in the deployment pipelines. This kind of environments will require experimentation to discover which workflows, tools and practices are best for them.

Acknowledgment

The work described in this article is partially funded by the ' Optimised Energy Efficient Design Platform for Refurbishment at District Level' (OptEEmAL) project, Grant Agreement Number 680676, 2015-2019, as part of the European Union's Horizon 2020 research and innovation programme.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the Optimised Energy Efficient Design Platform for Refurbishment at District Level (OptEEmAL) project, Grant Agreement Number 680676, 2015-2019, as part of the European Union's Horizon 2020 research and innovation programme.

ORCID

Iñaki Prieto http://orcid.org/0000-0002-8407-6023
Jose Luis Izkara http://orcid.org/0000-0001-5145-1985
Rubén Béjar http://orcid.org/0000-0001-7866-3793

References

Abbasi, S. and Malek, M.R., 2015. Design and modeling of a 3D volunteered geographic information with an interoperable description for fundamental components of a building. *Journal of Geomatics Science and Technology*, 4 (4), 15–28.

Alam, N., *et al*., 2013. Towards automatic validation and healing of CityGML models for geometric and semantic consistency. *In*: *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*. Germany: Copernicus GmbH, 27–29.

Allweyer, T., 2016. *BPMN 2.0: introduction to the standard for business process modeling*. Germany: BoD–Books on Demand.

Berg, A.M., 2015. *Jenkins continuous integration cookbook - second edition*. 2nd ed. UK: Packt Publishing Ltd.

Biljecki, F., *et al*., 2015a. Propagation of positional error in 3D GIS: estimation of the solar irradiation of building roofs. *International Journal of Geographical Information Science*, 29 (12), 2269–2294. doi:10.1080/13658816.2015.1073292

Biljecki, F., *et al*., 2016. The most common geometric and semantic errors in CityGML datasets. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 13–22. doi:10.5194/isprs-annals-IV-2-W1-13-2016

Biljecki, F., *et al*., 2015b. Applications of 3D city models: state of the art review. *ISPRS International Journal of Geo-Information*, 4 (4), 2842–2889. doi:10.3390/ijgi4042842

Budhathoki, N.R. and Haythornthwaite, C., 2012. Motivation for open collaboration: crowd and community models and the case of openstreetmap. *American Behavioral Scientist*, 57 (5), 548–575. doi:10.1177/0002764212469364

Cao, R., *et al*., 2017. 3D building roof reconstruction from airborne LiDAR point clouds: a framework based on a spatial database. *International Journal of Geographical Information Science*, 31 (7), 1–22.

Chacon, S. and Straub, B., 2014. *Pro Git. Section 5.1*. 2nd ed. Berkely, CA: Apress.

Chaturvedi, K., *et al*., 2015. Managing versions and history within semantic 3D city models for the next generation of CityGML. *In*: *Selected papers from the 3D GeoInfo 2015 Conference*, Kuala Lumpur, Malaysia. Switzerland: Springer.

Coors, V. and Krämer, M., 2011. Integrating quality management into a 3D geospatial server. *In*: *International archives of the photogrammetry, remote sensing and spatial information sciences - ISPRS archives*. Germany: Copernicus GmbH, 7–12.

Döllner, J., *et al*., 2006. The virtual 3D city model of Berlin-managing, integrating, and communicating complex urban information. *In*: *Proceedings of the 25th international symposium on urban data management UDMS 2006 in Aalborg, Denmark, 15-17 May 2006*.

Dyer, M., Gleeson, D., and Grey, T., 2017. Framework for collaborative urbanism. *In*: *Citizen empowerment and innovation in the data-rich city*. Switzerland: Springer, 19–30.

Gentle, A., 2015. *Continuous integration and delivery for documentation [online]*. Available from: https://opensource.com/business/15/7/continuous-integration-and-continuous-delivery-documentation

Goetz, M., 2012. Towards generating highly detailed 3D CityGML models from OpenStreetMap. *International Journal of Geographical Information Science*, 27 (5), 1–21.

Goetz, M. and Zipf, A., 2012 April. OpenStreetMap in 3D – detailed insights on the current situation in Germany. *In*: *City, proceedings of the AGILE 2012 international conference on geographic information science*, Avignon. Berlin Heidelberg: Springer-Verlag Berlin Heidelberg, 24–27.

Grey, T., Dyer, M., and Gleeson, D., 2017. Using big and small urban data for collaborative urbanism. *In*: *Citizen empowerment and innovation in the data-rich city*. Switzerland: Springer. 31–54.

Gröger, G. and Coors, V., 2011. Modeling guide for 3D objects. *SIG 3D Quality Working Group*. Available from: https://files.sig3d.org/file/ag-qualitaet/201311_SIG3D_Modeling_Guide_for_3D_Objects_Part_1.pdf

Gröger, G., *et al*., 2012. *OpenGIS city geography markup language (CityGML) encoding standard, version 2.0.0*. OGC Document No. 12-019.

Gröger, G., *et al*., 2005. Integrating versions, history and levels-of-detail within a 3D geodatabase. *In*: *Proc. of Int. Workshop on Next Generation City Models*. Bonn, Germany: EuroSDR publications.

Humble, J. and Farley, D., 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Continuous delivery. Boston: Pearson Education Inc.

Isikdag, U. and Zlatanova, S., 2010. Interactive modelling of buildings in Google Earth : a 3D tool for urban planning. *Developments in 3D Geo-Information Sciences*, 52–70.

ISO, 2010. *BS ISO 29481-1 : 2010 - Building information modelling - Information delivery manual - Part 1: methodology and format*. Switzerland: ISO, 34.

Kang, T.W. and Hong, C.H., 2015. IFC-CityGML LOD mapping automation based on multi-processing. *In*: *ISARC. Proceedings of the international symposium on automation and robotics in construction*, Canada. 1.

Ledoux, H., *et al*., 2013. Sessie 5: validation and repair of 2D & 3D data. *In*: *OSGeo. nl dag 2013*, 13 November 2013. Delft, The Netherlands: Delft University of Technology.

Limp, W.F., *et al*., 2010. Approaching 3D digital heritage data from a multi-technology, lifecycle perspective. *In*: *Proceedings of the 38th annual international conference on computer applications and quantitative methods in archaeology (CAA), Granada, Spain*. Oxford, England: BAR Publishing.

Malamboa, L. and Hahnb, M., 2010. LiDAR assisted CityGML creation. *In*: Franz-Josef Behr, A.P. Pradeepkumar, C.A. Beltrán Castañón, eds. *3rd summer applied geoinformatics for society and environment*. Stuttgart: STUTTGART Active Alumni Group.

Morel, M. and Gesquière, G., 2014. Managing temporal change of cities with CityGML. *Eurographics Workshop on Urban Data Modelling and Visualisation*, 37–42. Available from: https://dl.acm.org/citation.cfm?id=2855630

Ohori, K.A., *et al*., 2015. Modeling a 3D city model and its levels of detail as a true 4D model. *ISPRS International Journal Geo-Inf*, 4 (3), 1055–1075. doi:10.3390/ijgi4031055

Peters, R., Stoter, J., and Ledoux, H., 2014. 3D city modelling. *European Spatial Data Research*, 122. Available from: http://www.eurosdr.net/sites/default/files/images/inline/eduserv14_3dcitymodel_precourse.pdf

Prieto, I., *et al*., 2015. Sustainable refurbishment in urban districts through a web-based tool based on 3D city model. *Sustainable Places*, 2015, 31.

Scully, T., *et al*., 2015. 3drepo. io: building the next generation Web3D repository with AngularJS and X3DOM. *In*: *Proceedings of the 20th international conference on 3D web technology*. 235–243. New York, NY: ACM.

Sindram, M., *et al*., 2016. Voluminator 2.0 - speeding up the approximation of the volume of defective 3d building models. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, III (July), 12–19.

Vakharia, H., 2015. *Git, Docker, and continuous integration for TeX documents* [online]. Available from: https://opensource.com/business/15/12/git-docker-continuous-integration-tex-documents

Wagner, D. and Ledoux, H., 2016. *OGC CityGML quality interoperability experiment*. Wayland, MA: Open Geospatian Consortium.

Wagner, D., *et al*., 2013. Geometric-semantical consistency validation of CityGML models. *In:* J. Pouliot, S. Daniel, F. Hubert and A. Zamyadi, eds. *Lecture Notes in Geoinformation and Cartography*. Germany: Springer, 171–192.

Zhao, J., Stoter, J., and Ledoux, H., 2014. A framework for the automatic geometric repair of CityGML models. *Lecture Notes in Geoinformation and Cartography book series (LNGC)*, *Cartography from Pole to Pole*, 187–202.